Argonne NATIONAL LABORATORY

ESP Programming Models

Pavan Balaji, Jeff Hammond, Paul Hargrove, John Mellor-Crummey

U.S. DEPARTMENT OF ENERGY

Overall Project

- Focuses on four parts
  - Tools (performance, debugger)
  - Libraries (math libraries, etc.)
  - Programming Models (MPI, GA, UPC, CAF)
  - Other System Software (OS, I/O)

Programming Models

- Improvements to individual Models
  - MPI, Global Arrays, PGAS models (UPC, CAF)
  - Scalability improvements
- Interoperability between different models
  - Including external models (threading models)
    - MPI+threads
  - MPI + GA
  - Maybe even between MPI + PGAS languages (?)

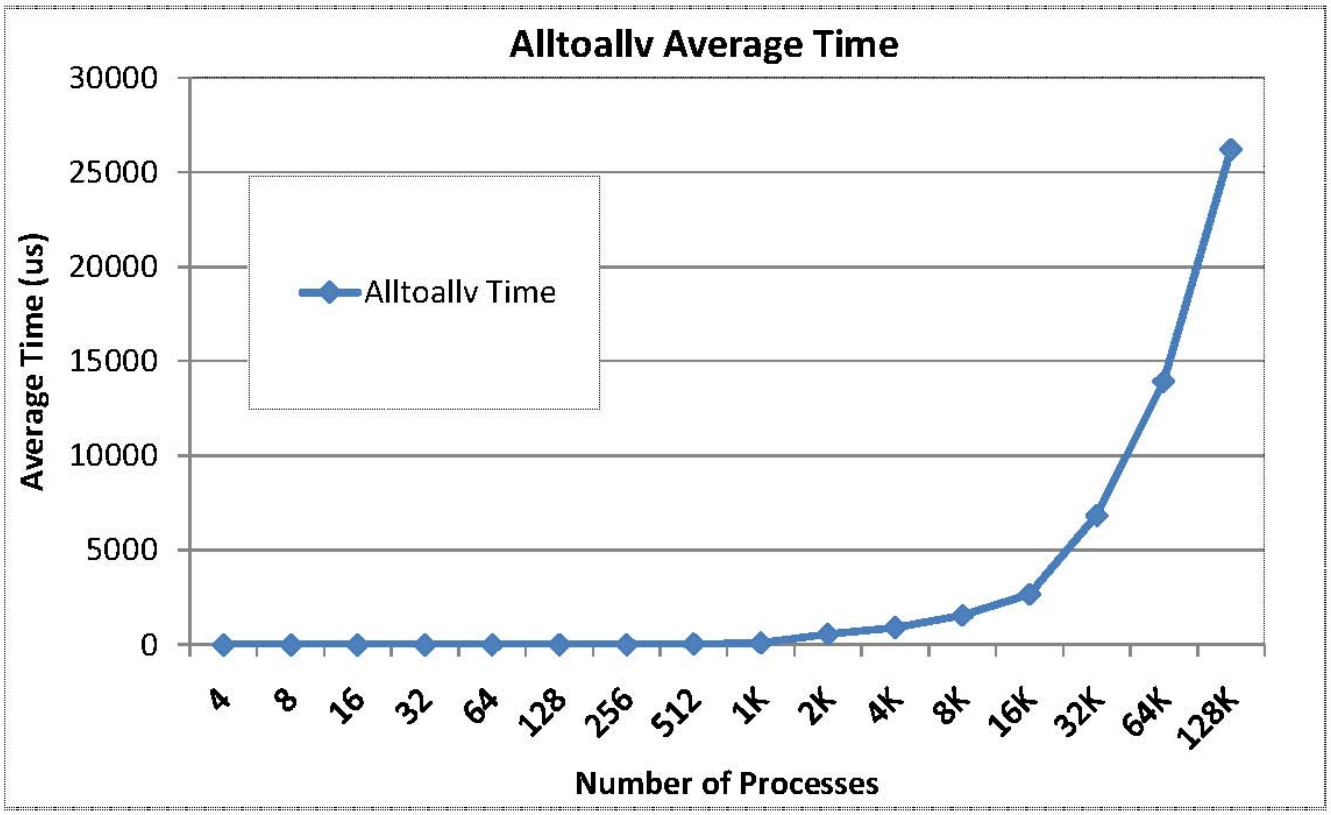
Plans for MPI

- IBM will provide MPI-2.2 capability
- Some of our enhancements will be improvements within the MPI-2.2 spec
  - Scalability issues
    - Memory scalability (constant memory scaling with increasing system scale is the holy grail)
    - Distributed memory data structures
  - Interoperability issues
    - Interoperating with threads (improvements for fine-grained locks)
- Some more enhancements from MPI-3

Factors Affecting MPI Scalability

- Performance and memory consumption
- A nonscalable MPI function is one whose time or memory consumption per process increase linearly (or worse) with the total number of processes (all else being equal)
- For example
  - If memory consumption of MPI_Comm_dup increases linearly with the no. of processes, it is not scalable
  - If time taken by MPI_Comm_spawn increases linearly or more with the no. of processes being spawned, it indicates a nonscalable implementation of the function
- Such examples need to be identified and fixed (in the specification and in implementations)
- The goal should be to use constructs that require only constant space per process

Zero-byte MPI_Alltoallv time on BG/P



- This is just the time to scan the parameter array to determine it is all 0 bytes. No communication performed.
- Enhancements in MPI-3 (sparse collectives) can help with this

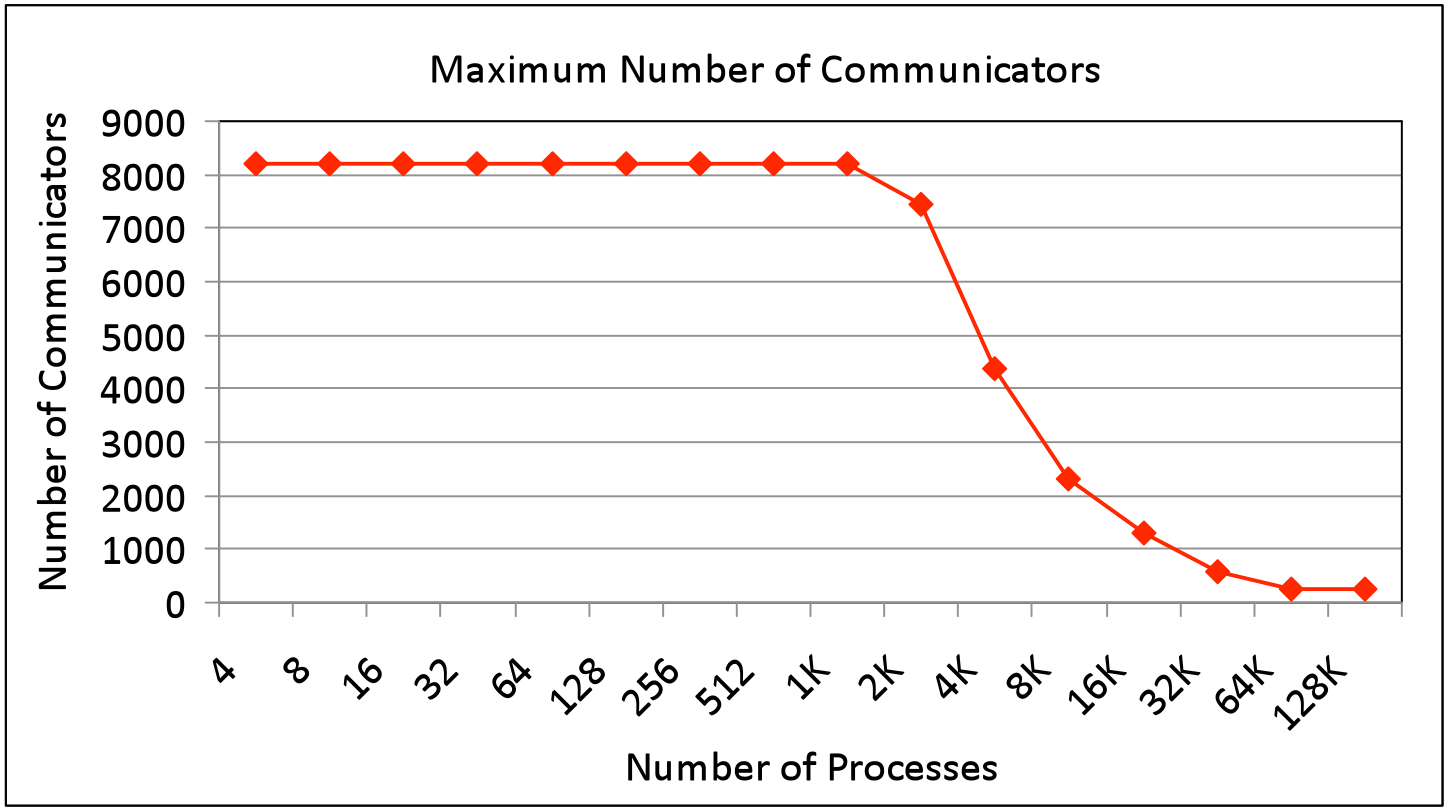
Communicator Memory Consumption

- NEK5000 is a well-known fluid dynamics code developed by Paul Fischer and colleagues at Argonne
- When they first tried to scale this code on the BG/P, it failed on as little as 8K processes because the MPI library ran out of communicator memory
- NEK5000 calls MPI_Comm_dup about 64 times (because it makes calls to libraries)
- 64 is not a large number, and, in any case, MPI_Comm_dup should not consume O(nprocs) memory (it doesn't in MPICH2)

- We ran an experiment to see what was going on…

Communicator Memory Consumption with original MPI on BG/P

- Run MPI_Comm_dup in a loop until it fails. Vary the no. of processes



What was going on --- and the fix

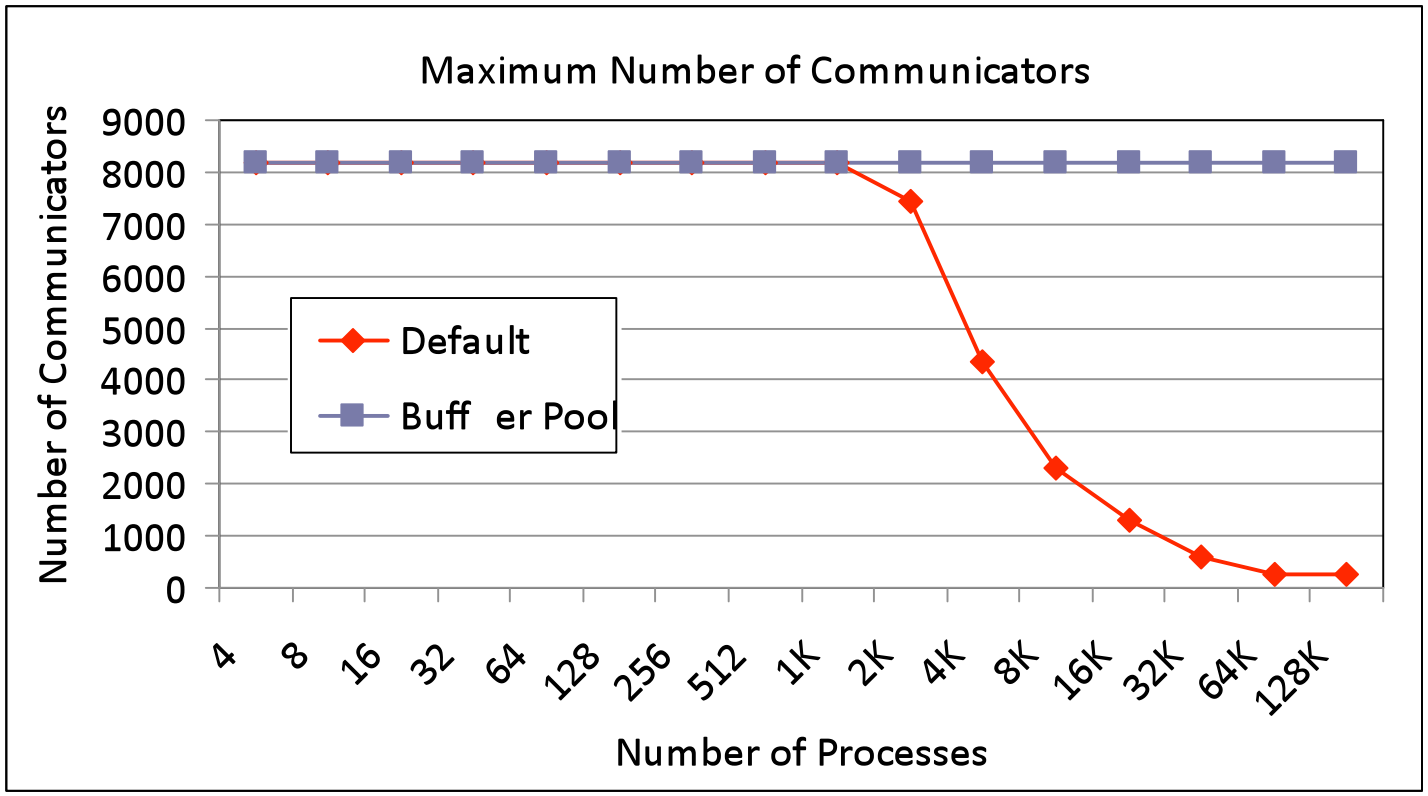
- The default MPI_Comm_dup in IBM's MPI was allocating memory to store process mapping info for optimizing future calls to collective communication (Alltoall)
- Allocated memory was growing linearly with system size
- One could disable the memory allocation with an environment variable, but that would also disable the collective optimizations

- On further investigation we found that they really only needed one buffer per thread instead of one buffer per new communicator

- Since there are only four threads on the BG/P, we fixed the problem by allocating a fixed buffer pool within MPI

- We provided IBM with a patch that fixed the problem

Communicator Memory Consumption Fixed



- NEK5000 code failed on BG/P at large scale because MPI ran out of communicator memory. We fixed the problem by using a fixed buffer pool within MPI and provided a patch to IBM.

MPI-3

- We will experiment with MPI-3 Remote Memory Access operations
  - Native support for NWChem/MADNESS kind of applications
  - Maybe even GFMC (through ADLB as well as for direct access accesses)
- MPI-3 Collective operations
- (Maybe) Hybrid programming enhancements with MPI +threads
  - Per-thread MPI ranks (threads as first-class citizens)